МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

НГТУ НЭТИ | Факультет прикладной математики и информатики

Кафедра прикладной математики

Практическое задание № 2

по дисциплине «Уравнения математической физики»

РЕШЕНИЕ ЭЛЛИПТИЧЕСКИХ КРАЕВЫХ ЗАДАЧ

МЕТОДОМ КОНЕЧНЫХ РАЗНОСТЕЙ

Бригада 1 ИСАКИН ДАНИИЛ

Группа ПМ-13 ВОСТРЕЦОВА ЕКАТЕРИНА

Вариант 5

Преподаватели ЗАДОРОЖНЫЙ АЛЕКСАНДР ГЕННАДЬЕВИЧ

ЛЕОНОВИЧ ДАРЬЯНА АЛЕКСАНДРОВНА

Новосибирск, 2024

1. Цель работы

Разработать программу решения нелинейной одномерной краевой задачи методом конечных элементов. Сравнить метод простой итерации и метод Ньютона для решения данной задачи.

2. Задание

1. Выполнить конечноэлементную аппроксимацию исходного уравнения в соответ-ствии с заданием. Получить формулы для вычисления компонент матрицы и вектора правой части для метода простой итерации.
2. Реализовать программу решения нелинейной задачи методом простой итерации с учетом следующих требований:
• язык программирования C++ или Фортран;
• предусмотреть возможность задания неравномерных сеток по пространству и по времени, разрывность параметров уравнения по подобластям, учет краевых условий;
• матрицу хранить в ленточном формате, для решения СЛАУ использовать методразложения;
• предусмотреть возможность использования параметра релаксации.
3. Выполнить линеаризацию нелинейной системы алгебраических уравнений с использованием метода Ньютона. Получить формулы для вычисления компонент линеаризованных матрицы и вектора правой части
4. Реализовать программу решения нелинейной задачи методом Ньютона.
5. Протестировать разработанные программы.

6. Исследовать реализованные методы на различных зависимостях коэффициента

от решения (или производной решения) в соответствии с заданием. На одних и тех же задачах сравнить по количеству итераций метод простой итерации и метод Ньютона. Исследовать скорость сходимости от параметра релаксации.
Вариант 5: Базисные функции линейные.

$$-div(\lambda(u)grad(u)) + \sigma\frac{du}{dt} = f$$

3. Анализ

Произведя временную аппроркимацию по двуслойной неявной схеме исходное уравнение примет вид:

$$-div(\lambda(u)grad(u) + \frac{\sigma}{\Delta t_s}u_s = f + \frac{\sigma}{\Delta t_s}u_{s-1}$$

В ходе конечноэлементной аппроксимации нелинейной начально-краевой задачи получается система нелинейных уравнений

$$\mathbf{A}(\mathbf{q_s})\mathbf{q_s} = \mathbf{b}(\mathbf{q_s})$$

у которой компоненты матрицы **A(qs)qs** и вектора правой части **b(qs)** вычисляются следующим образом:

$$A_{ij}(q_s) = \int_\Omega \lambda_s(u^h(q_s))grad\psi_i grad\psi_j d\Omega + \frac{1}{\Delta t_s}\int_\Omega \sigma_s(u^h(q_s))\psi_i\psi_j d\Omega + \int_{S_3}\beta_s(u^h(q_s))\psi_i\psi_j dS$$

$$b_i(q_s) = \int_\Omega f_s(u^h(q_s))\psi_i d\Omega + \frac{1}{\Delta t_s}\int_\Omega (u^h(q_s))(u^h(q_{s-1}))+$$

$$+\int_{S_2}\Theta_s(u^h(q_s))\psi_i dS + \int_{S_2}\beta_s(u^h(q_s))u_{\beta,s}(u^h(q_s))\psi_i dS$$

где

$$u^h(q_s) = \sum_k q_{k,s}\psi_k \qquad u^h(q_{s-1}) = \sum_k q_{k,s-1}\psi_k$$

$$G_{i,j} = \int_\Omega \lambda(u)grad\psi_i grad\psi_j d\Omega$$

$$G_{0,0} = \sum_{k=0}^{1}\int_\Omega \lambda_k\psi_k grad\psi_0 grad\psi_0 d\Omega =$$

$$= \int_\Omega \lambda_0\psi_0 grad\psi_0 grad\psi_0 d\Omega + \int_\Omega \lambda_1\psi_1 grad\psi_0 grad\psi_0 d\Omega =$$

$$= \frac{1}{h}\Big[\lambda_0\int_\Omega \psi_0 d\Omega + \lambda_1\int_\Omega \psi_1 d\Omega\Big] = \frac{1}{h}\Big[\lambda_0\int_0^1 \xi d\xi + \lambda_1\int_0^1 (1-\xi)d\xi\Big] =$$

$$= \frac{1}{h}\Big[\lambda_0\frac{\xi^2}{2}\Big|_0^1 + \lambda_1(\xi - \frac{\xi^2}{2})\Big|_0^1\Big] = \frac{\lambda_0+\lambda_1}{2h} = G_{1,1}$$

$$G_{0,1} = \sum_{k=0}^{1}\int_\Omega \lambda_k\psi_k grad\psi_1 grad\psi_1 d\Omega =$$

$$= \int_\Omega \lambda_0\psi_0 grad\psi_1 grad\psi_1 d\Omega + \int_\Omega \lambda_1\psi_1 grad\psi_1 grad\psi_1 d\Omega =$$

$$= -\frac{1}{h}\Big[\lambda_0\int_\Omega \psi_0 d\Omega + \lambda_1\int_\Omega \psi_1 d\Omega\Big] = -\frac{1}{h}\Big[\lambda_0\int_0^1 \xi d\xi + \lambda_1\int_0^1 (1-\xi)d\xi\Big] =$$

$$= -\frac{1}{h}\Big[\lambda_0\frac{\xi^2}{2}\Big|_0^1 + \lambda_1(\xi - \frac{\xi^2}{2})\Big|_0^1\Big] = -\frac{\lambda_0+\lambda_1}{2h} = G_{1,0}$$

$$G = \frac{\lambda_0+\lambda_1}{2h}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$$

$$M_{i,j} = \frac{\sigma}{\Delta t_s}\int_\Omega \psi_i\psi_j d\Omega$$

$$M_{0,0} = \frac{\sigma}{\Delta t_s}\int_\Omega \psi_0\psi_0 d\Omega = \frac{\sigma h}{\Delta t_s}\int_0^1 \xi^2 d\xi = \frac{\sigma h}{\Delta t_s}\frac{\xi^3}{3}\Big|_0^1 = \frac{\sigma h}{3\Delta t_s} = M_{1,1}$$

$$M_{0,1} = \frac{\sigma}{\Delta t_s}\int_\Omega \psi_0\psi_1 d\Omega = \frac{\sigma h}{\Delta t_s}\int_0^1 \xi(1-\xi) d\xi = \frac{\sigma h}{\Delta t_s}\Big(\frac{\xi^2}{2} - \frac{\xi^3}{3}\Big)\Big|_0^1 = \frac{\sigma h}{6\Delta t_s} = M_{1,0}$$

$$M = \frac{\sigma h}{6\Delta t_s}\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

$$b_i = \int_\Omega f_s\psi_i d\Omega + \frac{1}{\Delta t_s}\int_\Omega \sigma u^h_{q-1}\psi_i d\Omega \;\Big|\; u_{q-1}h = \sum_{k=0}^{1} q_{k,s-1}\psi_k$$

$$b_0 = \sum_{k=0}^{1}\int_\Omega f_k\psi_k\psi_0 d\Omega + \frac{\sigma}{\Delta t_s}\sum_{k=0}^{1}\int_\Omega q_{k,q-1}\psi_k\psi_0 d\Omega$$

$$= \Big[f_0\int_\Omega \psi_0\psi_0 d\Omega + f_1\int_\Omega \psi_1\psi_0 d\Omega\Big] + \frac{\sigma}{\Delta t_s}\Big[q_{0,s-1}\int_\Omega \psi_0\psi_0 d\Omega + q_{1,s-1}\int_\Omega \psi_1\psi_0 d\Omega\Big]$$

$$= h\Big[f_0\int_0^1 \xi^2 d\xi + f_1\int_0^1 (1-\xi)\xi d\xi\Big] + \frac{\sigma}{\Delta t_s}\Big[q_{0,s-1}\int_0^1 \xi^2 d\xi + q_{1,s-1}\int_0^1 (1-\xi)\xi d\xi\Big]$$

$$= h\Big[f_0\frac{\xi^3}{3}\Big|_0^1 + f_1\Big(\frac{\xi^2}{2} - \frac{\xi^3}{3}\Big)\Big|_0^1\Big] + \frac{\sigma}{\Delta t_s}\Big[q_{0,s-1}\frac{\xi^3}{3}\Big|_0^1 + q_{1,s-1}\Big(\frac{\xi^2}{2} - \frac{\xi^3}{3}\Big)\Big|_0^1\Big]$$

$$= h\Big[f_0\frac{1}{3} + f_1\frac{1}{6}\Big] + \frac{\sigma}{\Delta t_s}\Big[\frac{1}{3}q_{0,s-1} + \frac{1}{6}q_{1,s-1}\Big]$$

$$= \frac{h}{6}\Big[2f_0 + f_1\Big] + \frac{\sigma}{6\Delta t_s}\Big[2q_{0,s-1} + q_{1,s-1}\Big]$$

$$b_1 = \sum_{k=0}^{1}\int_\Omega f_k\psi_k\psi_1 d\Omega + \frac{\sigma}{\Delta t_s}\sum_{k=0}^{1}\int_\Omega q_{k,q-1}\psi_0\psi_1 d\Omega =$$

$$= \Big[f_0\int_\Omega \psi_0\psi_1 d\Omega + f_1\int_\Omega \psi_1\psi_1 d\Omega\Big] + \frac{\sigma}{\Delta t_s}\Big[q_{0,s-1}\int_\Omega \psi_0\psi_1 d\Omega + q_{1,s-1}\int_\Omega \psi_1\psi_1 d\Omega\Big] =$$

$$= h\Big[f_0\int_0^1 \xi(1-\xi) d\xi + f_1\int_0^1 (1-\xi)^2 d\xi\Big] + \frac{\sigma}{\Delta t_s}\Big[q_{0,s-1}\int_0^1 \xi(1-\xi) d\xi + q_{1,s-1}\int_0^1 (1-\xi)^2 d\xi\Big] =$$

$$= h\Big[f_0\Big(\frac{\xi^2}{2} - \frac{\xi^3}{3}\Big)\Big|_0^1 + f_1(1-\xi)^3\Big|_0^1\Big] + \frac{\sigma}{\Delta t_s}\Big[q_{0,s-1}\Big(\frac{\xi^2}{2} - \frac{\xi^3}{3}\Big)\Big|_0^1 + q_{1,s-1}(1-\xi)^3\Big|_0^1\Big] =$$

$$= \frac{h}{6}\Big[f_0\frac{1}{6} + f_1\frac{1}{3}\Big] + \frac{\sigma}{\Delta t_s}\Big[\frac{1}{6}q_{0,s-1} + \frac{1}{3}q_{1,s-1}\Big]$$

$$= \frac{h}{6}\Big[f_0 + 2f_1\Big] + \frac{\sigma}{6\Delta t_s}\Big[q_{0,s-1} + 2q_{1,s-1}\Big]$$

$$b = \frac{hx}{6}\begin{pmatrix} 2f_0 + f_1 \\ f_0 + 2f_1 \end{pmatrix} + \frac{\sigma}{6\Delta t_s}\begin{pmatrix} 2q_{0,s-1} + q_{1,s-1} \\ q_{0,s-1} + 2q_{1,s-1} \end{pmatrix}$$

---

4. Точность для разных функций u и λ

В ходе следующего исследования использовались следующие параметры:
ε = 1e − 7
σ=1
maxiter = 10000
Область пространства Ω = [0, 1]
Время задано на отрезке [0, 1]
Первоначальное число узлов 11, а конечных элементов 10

В таблице преведено количество итераций по нелинейности на последнем временном слое. В таблице приведены итерации на последнемм временном слое

lambda(u) = 1

u(x,t)	|| u(x,t)* - u(x,t) ||	Iteration
3x + t	1.303434e-08	2
2x^2 + t	3.651512e-03	2
x^3 + t	3.158513e-03	2
x^4 + t	4.147758e-03	2
exp(x) + t	1.530666e-03	2
3x + t	1.303434e-08	2
3x + t^2	9.043549e-03	2
3x + t^3	2.353030e-02	2
3x + exp(t)	1.086183e-02	2
3x + sin(t)	3.417052e-03	2
exp(x) + t^2	1.024960e-02	2
exp(x) + t^3	2.470912e-02	2
exp(x) + exp(t)	1.206169e-02	2
exp(t) + sin(t)	2.470657e-03	2

lambda(u) = u

u(x,t)	|| u(x,t)* - u(x,t) ||	Iteration
3x + t	1.193225e-08	2
2x^2 + t	2.151010e-03	8
x^3 + t	2.621328e-03	7
x^4 + t	8.481162e-03	7
exp(x) + t	1.147491e-03	6
3x + t	1.193225e-08	2
3x + t^2	4.122076e-03	6
3x + t^3	1.146758e-02	7
3x + exp(t)	2.875752e-03	5
3x + sin(t)	1.717295e-03	5
exp(x) + t^2	4.186961e-03	6
exp(x) + t^3	1.048465e-02	6
exp(x) + exp(t)	3.615697e-03	6
exp(t) + sin(t)	1.436147e-03	6

lambda(u) = u*u

u(x,t)	|| u(x,t)* - u(x,t) ||	Iteration
3x + t	1.613714e-07	13
2x^2 + t	7.830136e-03	13
x^3 + t	9.145729e-03	10
x^4 + t	2.590623e-02	10
exp(x) + t	1.158804e-03	9
3x + t	1.613714e-07	13
3x + t^2	2.022752e-03	14
3x + t^3	5.805964e-03	15
3x + exp(t)	7.332415e-04	9
3x + sin(t)	9.457542e-04	14
exp(x) + t^2	1.294761e-03	9
exp(x) + t^3	3.529498e-03	9
exp(x) + exp(t)	1.331907e-03	7
exp(t) + sin(t)	1.671499e-03	9

lambda(u) = u^3

u(x,t)	|| u(x,t)* - u(x,t) ||	Iteration
3x + t	1.333963e-07	23
2x^2 + t	2.860556e-02	19
x^3 + t	2.305492e-02	13
x^4 + t	6.414780e-02	14
exp(x) + t	2.460696e-03	11
3x + t	1.333963e-07	23
3x + t^2	1.090176e-03	23
3x + t^3	3.173479e-03	25
3x + exp(t)	1.913184e-04	13
3x + sin(t)	5.794735e-04	24
exp(x) + t^2	1.983732e-03	12
exp(x) + t^3	1.210300e-03	12
exp(x) + exp(t)	1.063126e-03	9
exp(t) + sin(t)	3.038160e-03	11

lambda(u) = exp(u)

u(x,t)	|| u(x,t)* - u(x,t) ||	Iteration
3x + t	4.980940e-05	16
2x^2 + t	8.503391e-03	12
x^3 + t	4.792654e-03	9
x^4 + t	1.443330e-02	9
exp(x) + t	2.797745e-03	10
3x + t	4.980940e-05	16
3x + t^2	9.423127e-04	16
3x + t^3	2.809399e-03	17
3x + exp(t)	1.804875e-04	16
3x + sin(t)	5.160900e-04	15
exp(x) + t^2	2.169186e-03	11
exp(x) + t^3	1.176447e-03	11
exp(x) + exp(t)	2.649043e-03	11
exp(t) + sin(t)	3.104545e-03	10

lambda(u) = u^4

u(x,t)	|| u(x,t)* - u(x,t) ||	Iteration
3x + t	1.231334e-07	36
2x^2 + t	1.234856e-01	32
x^3 + t	5.179322e-02	17
x^4 + t	1.599983e-01	20
exp(x) + t	5.006830e-03	14
3x + t	1.231334e-07	36
3x + t^2	6.336353e-04	37
3x + t^3	1.854730e-03	40
3x + exp(t)	5.110031e-05	17
3x + sin(t)	3.834758e-04	38
exp(x) + t^2	4.801635e-03	15
exp(x) + t^3	4.403118e-03	15
exp(x) + exp(t)	1.738079e-03	10
exp(t) + sin(t)	5.872785e-03	14

lambda(u) = 2 + sin(u)

u(x,t)	|| u(x,t)* - u(x,t) ||	Iteration
3x + t	1.002978e-05	8
2x^2 + t	4.737454e-03	6
x^3 + t	3.304153e-03	4
x^4 + t	4.426730e-03	4
exp(x) + t	2.133247e-03	7
3x + t	1.002978e-05	8
3x + t^2	3.730712e-03	8
3x + t^3	1.033199e-02	8
3x + exp(t)	8.796293e-03	8
3x + sin(t)	1.421441e-03	7
exp(x) + t^2	5.714441e-03	7
exp(x) + t^3	1.227563e-02	7
exp(x) + exp(t)	1.007523e-02	6
exp(t) + sin(t)	1.077934e-03	6

4.1. Вывод

Как видно из таблиц при отсутствие нелинейности итерационный процесс сходится за 2 итерации (расчет q1 и q2 и их сравнение). При этом если взять полином первой степени, то расчет происходит без погрешностей. В таблице погрешность обусловлена тем фактом, что правая часть расчитывается численно, что неизбежно приводит к накоплению погрешности машинной. При подстановке аналитического выражения правой части в уравнение погрешность порядка 10^{-16}.

5. Исследование на порядок сходимости.

Тест №1 (λ(u) = 1)

В ходе следующего исследования использовались следующие параметры:
ε = 1e − 7
σ=1
maxiter = 10000
Область пространства Ω = [0, 1]
Время задано на отрезке [0, 1]
Первоначальное число узлов 11, а конечных элементов 10
Для неравномерных сеток по времени и пространству коэффициент k=1.1
Функция λ(u) = 1
В таблицах приведено количество итераций на последнем временном слое.

lambda(u) = 1 u(x,t) = 3x + t

N	Nodes	Iteration	|| u(x,t)* - u(x,t) ||	log(|| * ||(h)/|| * ||(h/2))
1	11	2	1.303434e-08	0
2	21	2	1.246390e-08	6.456259e-02
3	41	2	3.858623e-09	1.691598e+00
4	81	2	1.057715e-09	1.867135e+00
5	161	2	1.519514e-09	-5.226584e-01

lambda(u) = 1 u(x,t) = 2x^2 + t

N	Nodes	Iteration	|| u(x,t)* - u(x,t) ||	log(|| * ||(h)/|| * ||(h/2))
1	11	2	3.651512e-03	0
2	21	2	9.128842e-04	1.999990e+00
3	41	2	2.282224e-04	1.999991e+00
4	81	2	5.705621e-05	1.999985e+00
5	161	2	1.426056e-05	2.000353e+00

lambda(u) = 1 u(x,t) = x^3 + t

N	Nodes	Iteration	|| u(x,t)* - u(x,t) ||	log(|| * ||(h)/|| * ||(h/2))
1	11	2	3.158513e-03	0
2	21	2	7.903353e-04	1.998709e+00
3	41	2	1.976298e-04	1.999664e+00
4	81	2	4.940984e-05	1.999930e+00
5	161	2	1.235478e-05	1.999729e+00

lambda(u) = 1 u(x,t) = x^4 + t

N	Nodes	Iteration	|| u(x,t)* - u(x,t) ||	log(|| * ||(h)/|| * ||(h/2))
1	11	2	4.147758e-03	0
2	21	2	1.039862e-03	1.995940e+00
3	41	2	2.601466e-04	1.998995e+00
4	81	2	6.504999e-05	1.999704e+00
5	161	2	1.626331e-05	1.999928e+00

lambda(u) = 1 u(x,t) = exp(x) + t

N	Nodes	Iteration	|| u(x,t)* - u(x,t) ||	log(|| * ||(h)/|| * ||(h/2))
1	11	2	1.530666e-03	0
2	21	2	3.826661e-04	2.000001e+00
3	41	2	9.565474e-05	2.000178e+00
4	81	2	2.390758e-05	2.000368e+00
5	161	2	5.977663e-06	1.999815e+00

Run Executable

lambda(u) = 1 u(x,t) = 3x + t

N	Nodes	Iteration	|| u(x,t)* - u(x,t) ||	log(|| * ||(h)/|| * ||(h/2))
1	11	2	1.303434e-08	0
2	21	2	1.246390e-08	6.456259e-02
3	41	2	3.858623e-09	1.691598e+00
4	81	2	1.057715e-09	1.867135e+00
5	161	2	1.519514e-09	-5.226584e-01

lambda(u) = 1 u(x,t) = 3x + t^2

N	Nodes	Iteration	|| u(x,t)* - u(x,t) ||	log(|| * ||(h)/|| * ||(h/2))
1	11	2	9.043549e-03	0
2	21	2	4.553380e-03	9.899514e-01
3	41	2	2.280652e-03	9.974912e-01
4	81	2	1.140835e-03	9.993563e-01
5	161	2	5.704858e-04	9.998272e-01

lambda(u) = 1 u(x,t) = 3x + t^3

N	Nodes	Iteration	|| u(x,t)* - u(x,t) ||	log(|| * ||(h)/|| * ||(h/2))
1	11	2	2.353030e-02	0
2	21	2	1.205747e-02	9.645924e-01
3	41	2	6.093998e-03	9.844661e-01
4	81	2	3.062298e-03	9.927743e-01
5	161	2	1.534845e-03	9.965221e-01

lambda(u) = 1 u(x,t) = exp(x) + t^3

N	Nodes	Iteration	|| u(x,t)* - u(x,t) ||	log(|| * ||(h)/|| * ||(h/2))
1	11	2	2.470912e-02	0
2	21	2	1.234980e-02	1.000555e+00
3	41	2	6.166776e-03	1.001900e+00
4	81	2	3.080443e-03	1.001379e+00
5	161	2	1.539375e-03	1.000793e+00

lambda(u) = 1 u(x,t) = exp(x) + exp(t)

N	Nodes	Iteration	|| u(x,t)* - u(x,t) ||	log(|| * ||(h)/|| * ||(h/2))
1	11	2	1.206169e-02	0
2	21	2	5.837335e-03	1.047051e+00
3	41	2	2.868321e-03	1.025103e+00
4	81	2	1.421335e-03	1.012960e+00
5	161	2	7.074441e-04	1.006559e+00

lambda(u) = 1 u(x,t) = exp(t) + sin(t)

N	Nodes	Iteration	|| u(x,t)* - u(x,t) ||	log(|| * ||(h)/|| * ||(h/2))
1	11	2	2.470657e-03	0
2	21	2	1.477942e-03	7.413047e-01
3	41	2	8.113462e-04	8.652003e-01
4	81	2	4.247614e-04	9.336652e-01
5	161	2	2.172464e-04	9.673200e-01

Тест №2 ($\lambda(u) = u^2 + 1$)

lambda(u) = u*u + 1 u(x,t) = 3x + t

N	Nodes	Iteration	|| u(x,t)* - u(x,t) ||	log(|| * ||(h)/|| * ||(h/2))
1	11	11	9.155494e-08	0
2	21	11	3.715434e-08	1.301107e+00
3	41	9	1.635214e-07	-2.137877e+00
4	81	8	5.153245e-08	1.665926e+00
5	161	6	2.625490e-07	-2.349034e+00

lambda(u) = u*u + 1 u(x,t) = 2x^2 + t

N	Nodes	Iteration	|| u(x,t)* - u(x,t) ||	log(|| * ||(h)/|| * ||(h/2))
1	11	11	3.890227e-03	0
2	21	10	9.798925e-04	1.989159e+00
3	41	9	2.454316e-04	1.997303e+00
4	81	7	6.154468e-05	1.995615e+00
5	161	6	1.507836e-05	2.029155e+00

lambda(u) = u*u + 1 u(x,t) = x^3 + t

N	Nodes	Iteration	|| u(x,t)* - u(x,t) ||	log(|| * ||(h)/|| * ||(h/2))
1	11	8	3.912039e-03	0
2	21	8	9.741860e-04	2.005651e+00
3	41	7	2.433090e-04	2.001408e+00
4	81	6	6.065667e-05	2.004051e+00
5	161	5	1.486094e-05	2.029141e+00

lambda(u) = u*u + 1 u(x,t) = x^4 + t

N	Nodes	Iteration	|| u(x,t)* - u(x,t) ||	log(|| * ||(h)/|| * ||(h/2))
1	11	9	1.214100e-02	0
2	21	8	2.986090e-03	2.023558e+00
3	41	7	7.433376e-04	2.006168e+00
4	81	7	1.856697e-04	2.001280e+00
5	161	6	4.631607e-05	2.003153e+00

lambda(u) = u*u + 1 u(x,t) = exp(x) + t

N	Nodes	Iteration	|| u(x,t)* - u(x,t) ||	log(|| * ||(h)/|| * ||(h/2))
1	11	8	1.066488e-03	0
2	21	8	2.669859e-04	1.998032e+00
3	41	7	6.679561e-05	1.998938e+00
4	81	6	1.669873e-05	2.000015e+00
5	161	5	4.054345e-06	2.042198e+00

lambda(u) = u*u + 1 u(x,t) = 3x + t

N	Nodes	Iteration	|| u(x,t)* - u(x,t) ||	log(|| * ||(h)/|| * ||(h/2))
1	11	11	9.155494e-08	0
2	21	11	3.715434e-08	1.301107e+00
3	41	9	1.635214e-07	-2.137877e+00
4	81	8	5.153245e-08	1.665926e+00
5	161	6	2.625490e-07	-2.349034e+00

lambda(u) = u*u + 1 u(x,t) = 3x + t^2

N	Nodes	Iteration	|| u(x,t)* - u(x,t) ||	log(|| * ||(h)/|| * ||(h/2))
1	11	12	1.538674e-03	0
2	21	11	7.837862e-04	9.731552e-01
3	41	10	3.937348e-04	9.932359e-01
4	81	8	1.971766e-04	9.977355e-01
5	161	7	9.845802e-05	1.001908e+00

lambda(u) = u*u + 1 u(x,t) = 3x + t^3

N	Nodes	Iteration	|| u(x,t)* - u(x,t) ||	log(|| * ||(h)/|| * ||(h/2))
1	11	12	4.411252e-03	0
2	21	11	2.294004e-03	9.433204e-01
3	41	10	1.164032e-03	9.787372e-01
4	81	8	5.857115e-04	9.908684e-01
5	161	7	2.934329e-04	9.971598e-01

lambda(u) = u*u + 1 u(x,t) = 3x + exp(t)

N	Nodes	Iteration	|| u(x,t)* - u(x,t) ||	log(|| * ||(h)/|| * ||(h/2))
1	11	9	6.858606e-04	0
2	21	8	3.524306e-04	9.605760e-01
3	41	8	1.781518e-04	9.842320e-01
4	81	7	8.943058e-05	9.942671e-01
5	161	6	4.462511e-05	1.002913e+00

lambda(u) = u*u + 1 u(x,t) = 3x + sin(t)

N	Nodes	Iteration	|| u(x,t)* - u(x,t) ||	log(|| * ||(h)/|| * ||(h/2))
1	11	11	6.966347e-04	0
2	21	10	3.596696e-04	9.537299e-01
3	41	9	1.816208e-04	9.857426e-01
4	81	7	9.154871e-05	9.883183e-01
5	161	6	4.584023e-05	9.979254e-01

lambda(u) = u*u + 1 u(x,t) = exp(x) + t^2

N	Nodes	Iteration	|| u(x,t)* - u(x,t) ||	log(|| * ||(h)/|| * ||(h/2))
1	11	8	1.316144e-03	0
2	21	8	5.685822e-04	1.210876e+00
3	41	7	2.838319e-04	1.002332e+00
4	81	6	1.447834e-04	9.711405e-01
5	161	6	7.345375e-05	9.789881e-01

lambda(u) = u*u + 1 u(x,t) = exp(x) + t^3

N	Nodes	Iteration	|| u(x,t)* - u(x,t) ||	log(|| * ||(h)/|| * ||(h/2))
1	11	9	3.215417e-03	0
2	21	8	1.677325e-03	9.388430e-01
3	41	7	8.634555e-04	9.579689e-01
4	81	7	4.386762e-04	9.769653e-01
5	161	6	2.212058e-04	9.877677e-01

lambda(u) = u*u + 1 u(x,t) = exp(x) + exp(t)

N	Nodes	Iteration	|| u(x,t)* - u(x,t) ||	log(|| * ||(h)/|| * ||(h/2))
1	11	7	1.340520e-03	0
2	21	7	4.524199e-04	1.567058e+00
3	41	6	1.830342e-04	1.305549e+00
4	81	6	8.350207e-05	1.132229e+00
5	161	5	4.048926e-05	1.044273e+00

lambda(u) = u*u + 1 u(x,t) = exp(t) + sin(t)

N	Nodes	Iteration	|| u(x,t)* - u(x,t) ||	log(|| * ||(h)/|| * ||(h/2))
1	11	8	1.393894e-03	0
2	21	7	4.560330e-04	1.611911e+00
3	41	7	1.756016e-04	1.376832e+00
4	81	6	7.696543e-05	1.190023e+00
5	161	5	3.599562e-05	1.096389e+00

5.1. Вывод

1) Из теста №1 видно, что при отсутствие нелинейности очевидно нет итераций по нелинейности. Так же из этого же теста видно, что порядок сходимости по пространству равен 2. В то время как по времени он 1. Соответственно при появлении нелинейного вхождения времени в решение порядок падает до 1.

2) Из Теста №2 видно, что при появлении нелинейности порядок сходимости по пространству практически не изменился, как и по времени. Что говорит о том, что линейная апроксимация довольно хорошо справляется с заданными функциями. Однако при большом значении производной данный метод будет довольно плохо работать, так как учета производной нет. Данную проблему можно решить при помощи метода Ньютона.

6. Исходный код программы

main.cpp

```
#include "FEM_1D.h"
#include <string>
#include <unistd.h>


function1D calcFirstDerivative(const function1D& f) {
	return [f](double x) -> double {
		const double h = 0.00001;
		return (f(x - 2 * h) - 8 * f(x - h) + 8 * f(x + h) - f(x + 2 * h) ) / (12 * h);
	};
}


function2D calcRightPart(const function1D& lambda, const function2D& u, double
sigma) {
	return [=](double x, double t) -> double {
		using namespace std::placeholders;
		auto duBydt = calcFirstDerivative(std::bind(u, x, _1));
		auto duBydx = calcFirstDerivative(std::bind(u, _1, t));
		auto lambda_grad = [=](double x, double t) -> double {
			return lambda(u(x, t)) * duBydx(x);
			//return lambda(duBydt(t)) * duBydx(x);
		};
		auto div = calcFirstDerivative(std::bind(lambda_grad, _1, t));
		return -div(x) + sigma * duBydt(t);
	};
}

/* Печатет табличку для исследования на сходимость */
void PrintTable(vector<string> &param_str, int CountOfDivide, function1D
&lambda_, function2D &u_)
{
	/* 0 = lambda_str
	   1 = u_str
	*/
	FEM fem;
	function2D f_ = calcRightPart(lambda_, u_, 1);
   fem.init(u_, f_, lambda_, 1, "Grid.txt", "TimeGrid.txt");

	pair<int, double> res_Prev = fem.solve();
```

```
        pair<int, double> res_Curr;
        printf("lambda(u) = %s  u(x,t) = %s", param_str[0].c_str(),
param_str[1].c_str());

printf("|-----------------------------------------------------------------------------------------
-----|\n");
        printf("|  N  |  Nodes  |   Iteration   |  || u(x,t)* - u(x,t) ||  |  log(|| * ||(h)/|| * ||(h/2))
|\n");

printf("|-----------------------------------------------------------------------------------------
-----|\n");
        printf("|%3d  |%6d   |%12d   |%22e    |%30e    |\n", 1, fem.getNodesCount(),
res_Prev.first, 0); // Первая строка в таблице

        for(int i = 2; i <= CountOfDivide; i++)
        {
                fem.DivideGridAndPrepareInternalParametrs(2);
                res_Curr = fem.solve();
                printf("|%3d  |%6d   |%12d   |%22e    |%30e    |\n", i,
fem.getNodesCount(), res_Curr.first, log2(res_Prev.second/res_Curr.second));


printf("|-----------------------------------------------------------------------------------------
-----|\n");
                res_Prev = res_Curr;
        }
        printf("\n\n\n");
}

int main()
{

   vector <function2D> u(14), f(14);
        u[0] = { [](double x, double t) -> double { return 3 * x + t; } };
        u[1] = { [](double x, double t) -> double { return 2 * x*x + t; } };
        u[2] = { [](double x, double t) -> double { return x * x*x + t; } };
        u[3] = { [](double x, double t) -> double { return x * x*x*x + t; } };
        u[4] = { [](double x, double t) -> double { return exp(x) + t; } };
        u[5] = { [](double x, double t) -> double { return 3 * x + t; } };
        u[6] = { [](double x, double t) -> double { return 3 * x + t * t; } };
        u[7] = { [](double x, double t) -> double { return 3 * x + t * t*t; } };
        u[8] = { [](double x, double t) -> double { return 3 * x + exp(t); } };
        u[9] = { [](double x, double t) -> double { return 3 * x + sin(t); } };
        u[10] = { [](double x, double t) -> double { return exp(x) + t * t; } };
        u[11] = { [](double x, double t) -> double { return exp(x) + t * t*t; } };
```

```
        u[12] = { [](double x, double t) -> double { return exp(x) + exp(t); } };
        u[13] = { [](double x, double t) -> double { return exp(x) + sin(t); } };

        vector <function1D>  lambda(8);
        lambda[0] = { [](double u) -> double {return 1; } };
        lambda[1] = { [](double u) -> double {return u; } };
        lambda[2] = { [](double u) -> double {return u * u; } };
        lambda[3] = { [](double u) -> double {return u * u + 1; } };
        lambda[4] = { [](double u) -> double {return u * u*u; } };
        lambda[5] = { [](double u) -> double {return u * u*u*u; } };
        lambda[6] = { [](double u) -> double {return exp(u); } };
        lambda[7] = { [](double u) -> double {return 2+sin(u); } };

    double sigma = 1;

        std::vector<std::string> lambda_str = {"1", "u", "u*u", "u*u + 1", "u^3", "u^4",
"exp(u)", "2 + sin(u)"};
        std::vector<std::string> u_str = {"3x + t", "2x^2 + t", "x^3 + t","x^4 + t",
"exp(x) + t", "3x + t", "3x + t^2", "3x + t^3", "3x + exp(t)", "3x + sin(t)", "exp(x) +
t^2", "exp(x) + t^3", "exp(x) + exp(t)", "exp(t) + sin(t)"};

        /* Генерация таблички для различных lambda */


        // for(int i = 0; i < lambda.size(); i++)
        // {
        //      printf("lambda(u) = %s\n", lambda_str[i].c_str());
        //      printf("|-----------------------------------------------------------------|\n");
        //      printf("|        u(x,t)         | || u(x,t)* - u(x,t) ||  |  Iteration  |\n");
        //      printf("|-----------------------------------------------------------------|\n");
        //      for(int j = 0; j < u.size(); j++)
        //      {
        //              f[j] = calcRightPart(lambda[i], u[j], sigma);
        //              FEM fem;
    //          fem.init(u[j], f[j], lambda[i], sigma, "Grid.txt", "TimeGrid.txt");
        //              auto res = fem.solve();
        //              printf("|%23s  |%20e      |%8d     |\n", u_str[j].c_str(), res.second,
res.first);
        //              printf("|-----------------------------------------------------------------|\
n");
        //              //sleep(1);
        //      }
        //      printf("\n\n\n");
        // }
```

```
	int Lambda_num = 0;
	/* Делаем проход по всем функциям в списке и фиксированной lambda */
	for(int k = 0; k < u.size(); k++)
	{

		FEM fem;
		f[k] = calcRightPart(lambda[Lambda_num], u[k], sigma);
		fem.init(u[k], f[k], lambda[Lambda_num], sigma, "Grid.txt",
"TimeGrid.txt");
		auto res_Prev = fem.solve();
		pair<int, double> res_Curr;
		printf("lambda(u) = %s  u(x,t) = %s\n",
lambda_str[Lambda_num].c_str(), u_str[k].c_str());


printf("|-----------------------------------------------------------------------------------------|\
n");
		printf("|  N  |  Nodes  |   Iteration   |  || u(x,t)* - u(x,t) ||  |  log(|| * ||(h)/|| * ||
(h/2))  |\n");


printf("|-----------------------------------------------------------------------------------------|\
n");
		printf("|%2d   |%6d   |%7d       |%18e       |%16s              |\n", 1,
fem.getNodesCount(), res_Prev.first, res_Prev.second, "0"); // Первая строка в
таблице


printf("|-----------------------------------------------------------------------------------------|\
n");

		for(int i = 2; i <= 5; i++)
		{
			fem.DivideGridAndPrepareInternalParametrs(2);
			res_Curr = fem.solve();
			printf("|%2d   |%6d   |%7d       |%18e       |       %13e          |\n", i,
fem.getNodesCount(), res_Curr.first, res_Curr.second,
log2(res_Prev.second/res_Curr.second) ); // Первая строка в таблице


printf("|-----------------------------------------------------------------------------------------|\
n");
			res_Prev = res_Curr;
		}
```

```
            printf("\n\n\n");
        }

    return 0;
}
```

head.h

```
#pragma once
#define _CRT_SECURE_NO_WARNINGS
#include <fstream>
#include <iostream>
#include <vector>
#include <string>
#include <iomanip>
#include <functional>
#include <cmath>

using namespace std;


typedef std::function<double(double)> function1D;
typedef std::function<double(double, double)> function2D;

typedef vector <double> vector1D;
typedef vector <vector <double>> matrix2D;


// Сравнение векторов
inline bool operator==(const vector1D& a, const vector1D& b) {
#ifdef _DEBUG
        if (a.size() != b.size())
                throw std::exception();
#endif
        for (int i = 0; i < a.size(); ++i)
                if (a[i] != b[i])
                        return false;

        return true;
}

// Сложение векторов
inline vector1D operator+(const vector1D& a, const vector1D& b) {
#ifdef _DEBUG
        if (a.size() != b.size())
                throw std::exception();
```

```
#endif
	vector1D result = a;
	for (int i = 0; i < b.size(); i++)
		result[i] += b[i];
	return result;
}
// Сложение матриц
inline matrix2D operator+(const matrix2D& a, const matrix2D& b) {
#ifdef _DEBUG
	if (a.size() != b.size())
		throw std::exception();
#endif
	matrix2D result = a;
	for (int i = 0; i < b.size(); i++)
		for (int j = 0; j < b.size(); j++)
			result[i][j] += b[i][j];
	return result;
}


// Деление матрицы на число
inline matrix2D operator/(const matrix2D& a, const double& b) {

	matrix2D result = a;
	for (int i = 0; i < a.size(); i++)
		for (int j = 0; j < a.size(); j++)
			result[i][j] /= b;
	return result;
}


// Вычитание векторов
inline vector1D operator-(const vector1D& a, const vector1D& b) {
#ifdef _DEBUG
	if (a.size() != b.size())
		throw std::exception();
#endif
	vector1D result = a;
	for (int i = 0; i < b.size(); i++)
		result[i] -= b[i];
	return result;
}
// Обратный знак вектора
inline vector1D operator-(const vector1D& a) {
	vector1D result = a;
```

```
	for (int i = 0; i < a.size(); i++)
		result[i] = -result[i];
	return result;
}



// Умножение матрицы на вектор
inline vector1D operator*(const matrix2D& a, const vector1D& b) {
	vector1D result = { 0.0, 0.0 };
	for (int i = 0; i < a.size(); i++)
		for (int j = 0; j < a.size(); j++)
			result[i] += a[i][j] * b[j];
	return result;
}



// Умножение на число
inline vector1D operator*(const vector1D& a, double b) {
	vector1D result = a;
	for (int i = 0; i < result.size(); i++)
		result[i] *= b;
	return result;
}
// Умножение на число
inline vector1D operator*(double b, const vector1D& a) {
	return operator*(a, b);
}



// Деление на число
inline vector1D operator/(const vector1D& a, double b) {
	vector1D result = a;
	for (int i = 0; i < result.size(); i++)
		result[i] /= b;
	return result;
}
// Деление на число
inline vector1D operator/(double b, const vector1D& a) {
	return operator/(a, b);
}
```

```
// Скалярное произведение
inline double operator*(const vector1D& a, const vector1D& b) {
#ifdef _DEBUG
	if (a.size() != b.size())
		throw std::exception();
#endif
	double sum = 0;
	for (int i = 0; i < a.size(); i++)
		sum += a[i] * b[i];
	return sum;
}


// Потоковый вывод вектора
inline std::ostream& operator<<(std::ostream& out, const vector1D& v) {
	for (int i = 0; i < v.size() - 1; ++i)
		out << v[i] << ", ";
	out << v.back();
	return out;
}
// Потоковый вывод матрицы
inline std::ostream& operator<<(std::ostream& out, const matrix2D& v) {
	for (int i = 0; i < v.size() - 1; ++i)
		out << v[i] << "  ";
	out << v.back();
	return out;
}


// Потоковый вывод вектора для TeX
inline void printTeXVector(std::ofstream &fout, const vector1D &v, int coefGrid) {
	fout << "$(";
	for (int i = 0; i < v.size() - 1; ++i)
		if (i % int(pow(2, coefGrid)) == 0)
			fout << v[i] << ", ";
	fout << v.back() << ")^T$";
}
```

Matrix.h

```
#ifndef MATRIX_H_
#define MATRIX_H_


#include <vector>
#include "head.h"
using namespace std;

struct Matrix
{
   /* Matrix */
   vector<int> ia;
   vector<double> di;
   vector<double> al;
   vector<double> au;
};


vector<double> MultAOnq(Matrix &matr, vector<double>& q);
double calcNormE(const vector1D &x);


#endif
```

Matrix.cpp

```
#include "Matrix.h"

vector<double> MultAOnq(Matrix &matr, vector<double>& q)
{
	vector1D tmp;
	tmp.resize(matr.di.size());

	if (matr.di.size() >= 2)
		tmp[0] = matr.di[0] * q[0] + matr.au[0] * q[1];

	if (matr.di.size() >= 3)
		for (size_t i = 1; i < matr.di.size() - 1; i++)
			tmp[i] = matr.al[i - 1] * q[i - 1] + matr.di[i] * q[i] + matr.au[i] * q[i
+ 1];

	int lIndex = matr.di.size() - 1;
	tmp[lIndex] = matr.al[lIndex - 1] * q[lIndex - 1] + matr.di[lIndex] * q[lIndex];
	return tmp;
}

double calcNormE(const vector1D &x) { return sqrt(x*x); }
```

Slau.h

```
#ifndef SLAU_H_
#define SLAU_H_

#include "Matrix.h"
#include <vector>

using namespace std;



struct Slau
{
   Matrix Matr;
   /* f */
   vector<double> f;
};

void LUDecomposition(Slau &slau);
void GausForward(Slau &slau,vector<double> &y);
```

```
void GausBack(Slau &slau, vector<double> &x);
void SolveSlau(Slau &slau ,vector<double>&x);
void TestSlau();


#endif
```

Slau.cpp

```
#include "Slau.h"

void LUDecomposition(Slau &slau)
{
   Matrix &Matr = slau.Matr;

   const int n = Matr.di.size();
   const int TriangMat = Matr.ia.size();

   for (int i = 0; i < n; i++)
   {
      int i0 = Matr.ia[i];
      int i1 = Matr.ia[i + 1];

      int j = i - (i1 - i0);
      double sd = 0;
      for (int m = i0; m < i1; m++, j++)
      {
         double sl = 0;
         double su = 0;

         int j0 = Matr.ia[j];
         int j1 = Matr.ia[j + 1];

         int mi = i0;
         int mj = j0;

         int kol_i = m - i0;
         int kol_j = j1 - j0;
         int kol_r = kol_i - kol_j;

         if (kol_r < 0)
            mj -= kol_r;
         else
            mi += kol_r;
```

```
            for (; mi < m; mi++, mj++)
            {
                sl += Matr.al[mi] * Matr.au[mj];
                su += Matr.au[mi] * Matr.al[mj];
            }

            Matr.au[m] = Matr.au[m] - su;
            Matr.al[m] = (Matr.al[m] - sl) / Matr.di[j];

            sd += Matr.al[m] * Matr.au[m];
        }
        Matr.di[i] = Matr.di[i] - sd;
    }
}

void GausForward(Slau &slau, vector<double> &y)
{
    Matrix &Matr = slau.Matr;
    int n = Matr.di.size();

    // Решение системы Ly = b
    for (int i = 0; i < n; i++)
    {
        auto &al = Matr.al;
        int i0 = Matr.ia[i];
        int i1 = Matr.ia[i + 1];
        double s = 0;
        int j = i - (i1 - i0);
        for (int k = i0; k < i1; k++, j++)
        {
            s += al[k] * y[j];
        }
        y[i] = slau.f[i] - s;
    }
}

void GausBack(Slau &slau, vector<double> &x)
{
    Matrix &Matr = slau.Matr;
    int n = Matr.di.size();
    //  Решение системы Ux = y
    for (int i = n - 1; i >= 0; i--)
    {
        double xi = x[i] / Matr.di[i];
        auto &au = Matr.au;
```

```
        int i0 = Matr.ia[i];
        int i1 = Matr.ia[i + 1];
        // int m = Matr.ai[i+1] - Matr.ai[i];
        int j = i - 1;
        for (int k = i1 - 1; k >= i0; k--, j--)
        {
            x[j] -= au[k] * xi;
        }
        x[i] = xi;
    }
}

void SolveSlau(Slau &slau, vector<double> &x)
{
    LUDecomposition(slau);
    GausForward(slau, x);
    GausBack(slau, x);
}

void TestSlau()
{
    vector<double> q;
    Slau slau;
    slau.Matr.di = { 1, 2.66667, 2.66667, 2.66667, 2.66667, 2.66667, 2.66667,
2.66667, 2.66667, 1 };
        slau.Matr.al = { -0.833333, -0.833333, -0.833333, -0.833333, -0.833333, -
0.833333, -0.833333, -0.833333, 0 };
        slau.Matr.au = { 0, -0.833333, -0.833333, -0.833333, -0.833333, -0.833333, -
0.833333, -0.833333, -0.833333 };
    slau.Matr.ia = { 0,0,1,2,3,4,5,6,7,8,9 };
        slau.f = { 1, 2.000008, 3.000012, 4.000016, 5.00002, 6.000024, 7.000028,
8.000032, 9.000036, 10 };
    q.resize(10, 0); // выходной вектор 1,2,3,4,5,6,7,8,9,10
        SolveSlau(slau, q);
        //cout << q << endl;
}
```

Grid_1D.h

```
#ifndef GRID_1D_H
#define GRID_1D_H
#include <vector>
#include <string>
#include <fstream>
using  namespace std;


/* Формат файла */
/*
   * Nx - Количество опорных узлов по оси x
   * x1 x2 ... xn
   * далее коэффиценты дробления сетки
   * n1 q1 n2 q2 ...

 */

class Grid_1D
{
   private:
   const double eps = 1e-10;
   ifstream fin;
   ofstream fout;
   struct DivideParamS
   {
      int num;     // количество интервалов на которое нужно разделить отрезок
      double coef; // Коэффициент растяжения или сжатия
   };
   int32_t Nx;
   vector<double> BaseGridX;
   vector<DivideParamS> DivideParam;
   vector<double> Grid;
   int stepCoef = 1; // Коэффициент с шагом для базовой сетки умножается при
дроблениях

   public:

   Grid_1D() = default;
   Grid_1D(const string &filename);
   void Load(const string &filenmae);

   void GenerateGrid();
```

```
    void DivideGrid(const int coef);
    void ReGenerateGrid();

    inline int32_t size() const { return (int32_t)Grid.size(); }
    inline double& operator[](const int32_t idx) { return Grid[(uint64_t)idx]; }


    void PrintGrid() const;

    Grid_1D& operator=(const Grid_1D &Grid_)
    {
        this->BaseGridX = Grid_.BaseGridX;
        this->Nx = Grid_.Nx;
        this->DivideParam = Grid_.DivideParam;
        this->Grid = Grid_.Grid;

        return *this;
    }

    ~Grid_1D() = default;
};



#endif
```

Grid_1D.cpp

```
#include "Grid_1D.h"
#include <iostream>
#include <cmath>

/* Private */

void Grid_1D::Load(const string &filename)
{
    fin.open(filename);

    if(!fin.is_open())
    {
        cout << "Файл можна????\n";
    }

    fin >> Nx;
    BaseGridX.resize((uint64_t)Nx);
```

```
    for(int32_t i = 0; i < Nx; i++)
        fin >> BaseGridX[(uint64_t)i];

    // Интервалов на 1 меньше чем опроных узлов
    DivideParam.resize(uint64_t(Nx-1));
    for(int32_t i = 0; i < Nx-1; i++)
        fin >> DivideParam[(uint64_t)i].num >> DivideParam[(uint64_t)i].coef;

    //cout << "File is load done\n";
    fin.close();
}

/* Public */

Grid_1D::Grid_1D(const string &filename)
{
    Load(filename);

    /* Рачет общего числа узлов */
    int32_t GlobalNx = 0;
    for(int32_t i = 0; i < Nx-1; i++)
        GlobalNx+=DivideParam[(uint64_t)i].num;

    GlobalNx++;
    Grid.resize((uint64_t)GlobalNx);
}

void Grid_1D::GenerateGrid()
{
     struct SettingForDivide
    {
        double step; // Шаг на отрезке
        double coef; // Коэффициент увеличения шага
        int num;     // Количество интервалов идем то num-1 и потом явно вставляем
элемент
    };

    /* Расчитываем шаг для сетки  */
    /*
        @param
        int j - Номер элемента в массиве
        double left - левая грани отрезка
        double right - правая граница отрезка
```

```
      ret: SettingForDivide -  структура с вычесленными параметрами деления сетки
   */
   auto CalcSettingForDivide = [&](int j, double left, double right) -> SettingForDivide
   {
      SettingForDivide res;
      int num = DivideParam[j].num;
      double coef = DivideParam[j].coef;

      if (coef > 1.0)
      {
         double coefStep = 1.0 + (coef * (std::pow(coef, num - 1) - 1.0)) / (coef - 1.0);

         res.step = (right - left) / coefStep;
      }
      else
      {
         res.step = (right - left) / num;
      }

      //  Убираем погрешность
      if (std::abs(res.step) < eps)
         res.step = 0.0;

      res.num = num;
      res.coef = coef;
      return res;
   };

   /* Генерация разбиения по X  с учетом разбиения */
   /*
      @param
      SettingForDivide &param - параметр разбиения
      double left - левая граница отрезка
      double right - правая граница отрезка
      double *Line - генерируемый массив
      int &idx - индекс в массиве на какую позицию ставить элемент
   */
   auto GenerateDivide = [](SettingForDivide &param, double left, double right, vector<double>& Line, int &idx) -> void
   {
      int num = param.num;
      double coef = param.coef;
      double step = param.step;
```

```
        Line[idx] = left;
        idx++;
        double ak = left;
        for (int k = 0; k < num - 1; k++)
        {
            ak = ak + step * std::pow(coef, k);
            Line[idx] = ak;
            idx++;
        }
        Line[idx] = right;
    };

    int idx = 0;
    for(int32_t j = 0; j < Nx-1; j++)
    {
        double left = BaseGridX[j];
        double right = BaseGridX[j+1];
        SettingForDivide param = CalcSettingForDivide(j, left, right);
        GenerateDivide(param, left, right, Grid, idx);
    }

}

void Grid_1D::DivideGrid(const int coef)
{
    for(uint64_t i = 0; i < DivideParam.size(); i++)
    {
        DivideParam[i].num *= static_cast<double>(coef);
        DivideParam[i].coef = pow(DivideParam[i].coef,
1.0/(static_cast<double>(coef)));
    }
    stepCoef *= coef;
}

void Grid_1D::ReGenerateGrid()
{
    Grid.clear(); // Очистка сетки

    /* Рачет общего числа узлов */
    int32_t GlobalNx = 0;
    for(int32_t i = 0; i < Nx-1; i++)
        GlobalNx+=DivideParam[(uint64_t)i].num;

    GlobalNx++;
```

```
    Grid.resize((uint64_t)GlobalNx);
    GenerateGrid(); // Перегенерация сетки
}


void Grid_1D::PrintGrid() const
{
    cout << "Print 1D Grid: \n";
    for(int32_t i = 0; i < Grid.size(); i++)
    {
        cout << Grid[i] << " ";
    }
    cout << "\n";
}
```

FEM_1D.h

```
#ifndef FEM_H_
#define FEM_H_


#include "Grid_1D.h"
#include "Slau.h"
#include <string>
#include <fstream>
#include <fstream>
#include <functional>
#include <cmath>

using namespace std;


typedef function<double (double, double)> function2D;
typedef function<double (double)> function1D;

class FEM
{
    private:
    Grid_1D Grid;
    Grid_1D TimeGrid;
    Slau slau; // Она же глобальная матрицы и вектор правой части
    vector<double> q, qPrev;
    vector<double> qExact; // Вектор на прошлой итерации по нелинейности
    vector<vector<double>> Q; // Общее решение по временным слоям

    double lambda0, lambda1;
```

```
    double sigma;
    double t;
    double dt;
    const int maxiter = 1000; // Максимальное количество итераций
    double eps = 1e-7;
    double delta = 1e-7;

    int StepCoef = 1; // Шаговый коэффициент по пространству
    int TimeCoef = 1; // Шаговый коэффициент по времени

    function2D f, u;
    function1D lambda;

    void GenerateProfile();
    void buildGlobalMatrixA(double _dt);
        void buildGlobalVectorb();
        void printGlobalMatrixA();
        void printGlobalVectorb();

        void buildLocalMatrixG(int elemNumber);
        void buildLocalMatrixM(int elemNumber);
        void buildLocalmatrixA(int elemNumber);
        void buildLocalVectorf(int elemNumber);
    bool shouldCalc(int i);

    double calcNormAtMainNodes()
    {
       double res = 0;
                auto normsub = [&](double x, double t)
       {
          return pow(u(x,t) - CalculateU(x,t), 2.0);
       };

       /* Расчет интеграла. Берем шаг h = 0.002 и пройдемся по отрезку и
вычислим интеграл */
       double h = 0.0002;
       double start = Grid[0];
       int32_t N = int32_t((Grid[Grid.size()-1] - Grid[0])/h);

       for(int32_t i = 0; i < N; i++)
       {
          double a = start + i*h;
          double b = a + h;
          double arg = (b+a)/2.0;
          //cout << "[ " << a << "; " << b << "]\n";
```

```
          //cout << "hx = " << h  << " arg = " << arg << " normsub() = " <<
normsub(arg, 1.0) << "\n";
          res += h*normsub(arg, 1.0);
       }

       return sqrt(res);
         }


   vector<vector<double>>  GLocal ,MLocal, ALocal;
        vector<double> bLocal;

   public:

   FEM() = default;

   void init(const function2D &_u, const function2D &_f, const function1D
&_lambda, double _sigma, const string &Grid, const string &TimeGrid);
        pair<int, double> solve();
        inline int getNodesCount() { return Grid.size(); }
   void DivideGridAndPrepareInternalParametrs(const int32_t coef);

   double CalculateU(double x, double t);

};


#endif
```

FEM_1D.cpp

```
#include "FEM_1D.h"
#include "head.h"
/* Private */

bool FEM::shouldCalc(int i)
{
   // Выход по макисмальному количеству итераций
   if(i > maxiter)
   {
      return false;
   }

        /* Выход по измененинию вектора решения */
        if(calcNormE(q-qExact)/calcNormE(q) < delta)
```

```
        {
                return false;
        }

    // Вывод по невязке
        //cout << "Non-repan: " << calcNormE(MultAOnq(slau.Matr, q) -
slau.f)/calcNormE(slau.f) << "\n";
    if( calcNormE(MultAOnq(slau.Matr, q) - slau.f)/calcNormE(slau.f) < eps )
    {
       return false;
    }

    return true;
}

void FEM::GenerateProfile()
{
    int n = Grid.size();

    slau.Matr.ia[0] = 0;

    for(int i = 1; i < n+1; i++)
       slau.Matr.ia[i] = i-1;

}

void FEM::buildGlobalMatrixA(double _dt)
{
        int nodesCount = Grid.size();
    int finiteElementsCount = nodesCount-1;
    dt = _dt;
        auto &di = slau.Matr.di;
    auto &au = slau.Matr.au;
    auto &al = slau.Matr.al;
        di.clear();
        au.clear();
        al.clear();

        di.resize(nodesCount, 0);
        al.resize(nodesCount - 1, 0);
        au.resize(nodesCount - 1, 0);

    for (size_t elemNumber = 0; elemNumber < finiteElementsCount; elemNumber++)
        {
                buildLocalmatrixA(elemNumber);
```

```
            //cout << ALocal << endl;
            di[elemNumber] += ALocal[0][0];          au[elemNumber] +=
ALocal[0][1];
            al[elemNumber] += ALocal[1][0];          di[elemNumber + 1] +=
ALocal[1][1];
        }

        // Первые краевые условия
        di[0] = 1;
        au[0] = 0;
        di[nodesCount - 1] = 1;
        al[al.size() - 1] = 0;
}

void FEM::buildGlobalVectorb()
{
   auto &f = slau.f;
   int nodesCount = Grid.size();
   int finiteElementsCount = nodesCount-1;

   f.clear();
        f.resize(nodesCount, 0);

        for (size_t elemNumber = 0; elemNumber < finiteElementsCount;
elemNumber++)
        {
            buildLocalVectorf(elemNumber);
            f[elemNumber] += bLocal[0];
            f[elemNumber + 1] += bLocal[1];
        }

        f[0] = u(Grid[0], t);
        f[nodesCount - 1] = u(Grid[nodesCount - 1], t);
}

void FEM::printGlobalMatrixA()
{

}

void FEM::printGlobalVectorb()
{

}
```

```
void FEM::buildLocalMatrixG(int elemNumber)
{
   double hx = Grid[elemNumber+1] - Grid[elemNumber];
        lambda0 = lambda(q[elemNumber]);
        lambda1 = lambda(q[elemNumber + 1]);
        double numerator = (lambda0 + lambda1) /(2.0 * hx);
        GLocal[0][0] = GLocal[1][1] = numerator;
        GLocal[0][1] = GLocal[1][0] = -numerator;
}

void FEM::buildLocalMatrixM(int elemNumber)
{
   double hx = Grid[elemNumber+1] - Grid[elemNumber];
        double numerator = (sigma * hx) / (6 * dt);
        MLocal[0][0] = MLocal[1][1] = 2 * numerator;
        MLocal[0][1] = MLocal[1][0] = numerator;
}

void FEM::buildLocalmatrixA(int elemNumber)
{
        ALocal = GLocal = MLocal = { {0,0}, {0,0} };
        buildLocalMatrixG(elemNumber);
        buildLocalMatrixM(elemNumber);
        for (size_t i = 0; i < 2; i++)
        {
                for (size_t j = 0; j < 2; j++)
                {
                        ALocal[i][j] = GLocal[i][j] + MLocal[i][j];
                }
        }
}

void FEM::buildLocalVectorf(int elemNumber)
{
   double hx = Grid[elemNumber+1] - Grid[elemNumber];
        bLocal = { 0, 0 };
        bLocal[0] = hx * (2.0 * f(Grid[elemNumber], t) + f(Grid[elemNumber + 1],
t)) / 6.0
                + sigma *hx * (2.0 * qPrev[elemNumber] + qPrev[elemNumber + 1]) /
(6.0 * dt);
        bLocal[1] = hx * (f(Grid[elemNumber], t) + 2.0 * f(Grid[elemNumber + 1],
t)) / 6.0
                + sigma * hx *(qPrev[elemNumber] + 2.0 * qPrev[elemNumber + 1]) /
(6.0 * dt);
```

```
}

/* Public */
void FEM::init(const function2D &_u, const function2D &_f, const function1D
&_lambda, double _sigma, const string &Grid_, const string &TimeGrid_)
{
   Grid = Grid_1D(Grid_);
   TimeGrid = Grid_1D(TimeGrid_);
        Grid.GenerateGrid();
        TimeGrid.GenerateGrid();

   u = _u;
        f = _f;
        lambda = _lambda;
        sigma = _sigma;

   /* Allocate memory for matrix and right part */
   int n = Grid.size();
   slau.Matr.di.resize(n);
   slau.Matr.au.resize(n-1);
   slau.Matr.al.resize(n-1);
   slau.Matr.ia.resize(n+1);
   slau.f.resize(n);
   /* Генерация профиля матрицы она имеет 3-х диагональную структуру */
   GenerateProfile();

   /* Память под вектора решений */
   q.resize(n);
   qPrev.resize(n);
        qExact.resize(n);
        Q.resize(TimeGrid.size());

   /* Память под локальные матрицы  */
   GLocal = vector(2, vector<double>(2));
   MLocal = vector(2, vector<double>(2));
   ALocal = vector(2, vector<double>(2));

}

void FEM::DivideGridAndPrepareInternalParametrs(const int32_t coef)
{
        /* Дробим сетку */
        Grid.DivideGrid(coef);
        Grid.ReGenerateGrid();
```

```
        TimeGrid.DivideGrid(coef);
        TimeGrid.ReGenerateGrid();

        slau.Matr.di.clear();
        slau.Matr.al.clear();
        slau.Matr.au.clear();
        slau.Matr.ia.clear();
        slau.f.clear();
        q.clear();
        qPrev.clear();
        qExact.clear();
        Q.clear();

         /* Allocate memory for matrix and right part */
    int n = Grid.size();
    slau.Matr.di.resize(n);
    slau.Matr.au.resize(n-1);
    slau.Matr.al.resize(n-1);
    slau.Matr.ia.resize(n+1);
    slau.f.resize(n);
    /* Генерация профиля матрицы она имеет 3-х диагональную структуру */
    GenerateProfile();

    /* Память под вектора решений */
    q.resize(n);
    qPrev.resize(n);
        qExact.resize(n);
        Q.resize(TimeGrid.size());
}

pair<int, double> FEM::solve()
{
        // Задаём начальные условия
        int n = Grid.size();
        //q.resize(n, 0);
        //qPrev.resize(n, 0);


        for (size_t i = 0; i < n; i++)
                qPrev[i] = u(Grid[i], TimeGrid[0]);
        //qPrev = qExact; // Прошлый временной слой его трогать нельзя он
прошлый и посчитан идеально

        Q[0] = qPrev;
        //cout << "QTrue: [" << qPrev << "]\n";
```

```
	int count = 0;
	// Решаем в каждый момент временной сетки
	double sumNormQ = 0;
	for (size_t i = 1; i < TimeGrid.size(); i++)
	{
		count = 0; // Обнулили счеткик итераций

		dt = TimeGrid[i] - TimeGrid[i - 1];
		t = TimeGrid[i];

		/* Производим первую итерацию q = q1 - первая итерация по нелинейности на новом временном слое */
		buildGlobalMatrixA(dt);
		buildGlobalVectorb();
		SolveSlau(slau, q);
		count++; // Итеарция прошла

		/* Сначала делаем еще одну итерацию по нелинейности и если нет падения погрешности, то заканчиваем итерации */
		do {
			qExact = q; // Сохранили векор после итерации по нелинейности это первый вектор посчитанный потом он будет меняться во время итераций по нелинейности

			buildGlobalMatrixA(dt);
			buildGlobalVectorb();
			SolveSlau(slau, q); // Расчитали новый q = q2 и.т.д

			count++; // Итеарция прошла

			//cout << "time: " << t << " q = [" << q << "]\n";
			//cout << "time: " << t << " qPrev = [" << qPrev << "]\n";

		} while (shouldCalc(count));

		qPrev = qExact; //  Новый временной слой присвоили
		//cout << "Time = " << t <<" Total iteration = " << count << "\n";
		Q[i] = q; // Заносим в решение очередной временной слой
	}

	return make_pair(count, calcNormAtMainNodes()); // в конце возвращаем количество итераций по нелинейности для последнего временного слоя и погрешность то же для последнего
}
```

```
double FEM::CalculateU(double x, double t)
{
	double res = 0;
	// Пусть мы берем последний временной слой пока что
	vector<double>& TmpQ = Q[Q.size()-1];

	/* Определим отрезок в котором будем расчитвыать значение функции */
	int32_t NumElement = 0;
	for(; NumElement < Grid.size()-1; NumElement++)
	{
		if(Grid[NumElement] <= x && Grid[NumElement+1] >= x) break;
	}

	double xm = Grid[NumElement];
	double xm1 = Grid[NumElement+1];

	auto Psi1 = [&](double x) { return (xm1 - x)/(xm1-xm); };
	auto Psi2 = [&](double x) { return (x-xm)/(xm1-xm); };

	return TmpQ[NumElement]*Psi1(x) + TmpQ[NumElement+1]*Psi2(x);
}
```